Deploying to the Cloud

Today we'll deploy your Node.js server application on a machine running in the AWS cloud!

- Register for AWS
- Go to the service EC2
- Launch New Instance
- Check the "Free-tier Only" box in the search and search for Ubuntu
- Select Ubuntu Server 18.04 LTS (HVM) which is a Linux Operating System

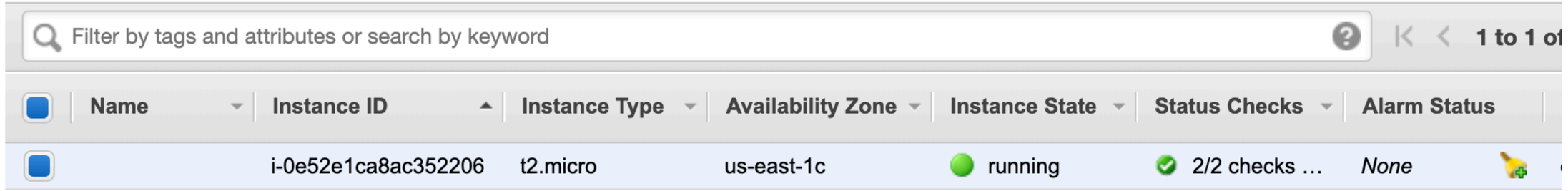
Instance Configuration

- General Purpose t2.micro (Free-tier Eligible)
- Next: Configure Instance Details (Accept Defaults)
- Next: Add Storage (Accept Defaults)
- Next: Add Tags (Accept Defaults)
- Next: Configure Security Group
  - Add 2 rules and from the Type drop down select HTTP and HTTPS
  - Add 1 more custom TCP rule for port 1234 and accept sources 0.0.0.0/0, ::/0
  - This opens up firewall rules that allow us to run HTTP and HTTPS on server
- Review and Launch, Launch!

Key Pairs

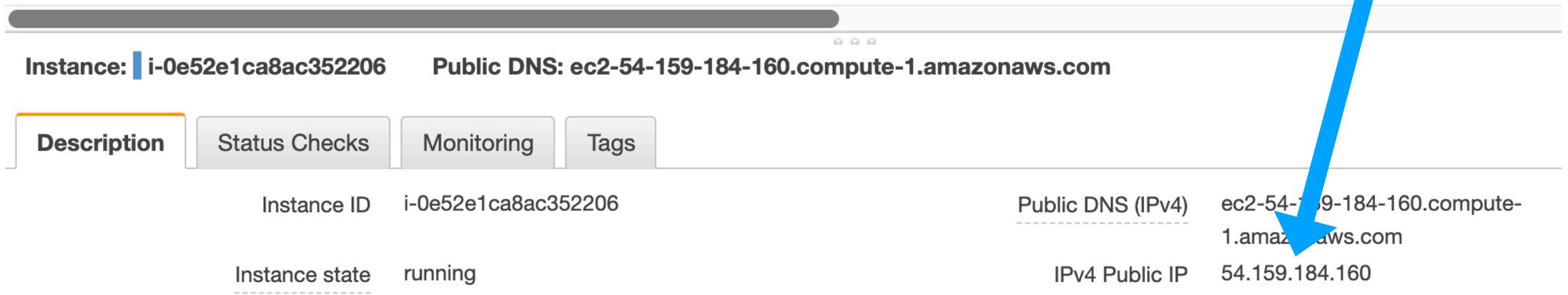
- Once the server launches, you need to be able to log into it!
- Amazon EC2 does this by installing a cryptographically secure key pair to authenticate you with over SSH.
- You should create a new Key Pair name ("COMP426-2019") and download it.
- You'll download a .pem file that we'll use in order to log into the server.
- From the "Launch Success" screen, you'll see a link to an Instance ID that looks something like: i-0e52e1ca8ac352206 - click it to see the details of your server

The Instances Screen



Your Servers

A Server's IP Address



SSH'ing with a PEM File

- Let's setup a directory in your user's $HOME directory for SSH files
  - $ mkdir -p ~/.ssh
- Move the .pem file to it
  - $ mv ~/Downloads/COMP426-2019.pem ~/.ssh
- (Mac Only) Set more restrictive file permissions
  - $ chmod 0400 ~/.ssh/COMP426-2019.pem
- Try SSH'ing in and hopefully you can login!
  - $ ssh -i ~/.ssh/COMP426-2019.pem ubuntu@<your-server-ip>

Update Your Server's "App Store" and Upgrade Preinstalled Operating System Programs

- The way to think of a Linux package manager like "aptitude" on Ubuntu is like an app store for developers and systems admins that is run at the command line

- The operating system keeps a local cache of package versions and it's best to update that cache because it could be out of date:

  - $ sudo apt-get update

- There may also be security and bug fixes to programs since the latest release of the operating system. It's worth going ahead and upgrading:

  - $ sudo apt-get upgrade

Let's Generate an SSH Key to Identify the Server on GitHub

- Generate an RSA Key-Pair:
  - $ ssh-keygen -t rsa -b 4096
  - Press enter to accept defaults without passphrase
- Print out the generated public key:
  - $ cat ~/.ssh/id_rsa.pub
- Copy and paste that resulting text as an accepted key on your GitHub account
  - Click on your User Icon
  - Go to Settings
  - SSH & GPG Keys
  - New SSH Key. Title: 426 Cloud Server, Key: Paste

Clone Your Project Repo

- Go to your project repo.
- Clone or Download "Use SSH"
- Copy the link that starts with git@github.com:cph426-2019/travel-notes-
- In the AWS instance terminal, run:
  - $ git clone <paste>
- Change directory into the repository: $ cd travel-notes-...
- Fetch all branches of repository: $ git fetch --all
- Checkout the server-side branch: $ git checkout server-side

Install Node.js 10 LTS

- The Ubuntu Server's default version of Node.js is 8 and we're using 10
- We need to add a custom package archive from NodeSource
  - Following: https://github.com/nodesource/distributions/blob/master/README.md
- Commands:
  - $ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
  - $ sudo apt-get install -y nodejs
  - $ node -v # You should see v10.16.0
- Then, try installing and running your server-side app!
  - $ npm install
  - $ npm run build
  - $ npm start # Once you see "Listening on port 1234" - navigate to <your-server-ip>:1234
- Ctrl+C to quit once you've got it working

Add a Production Script

- When deploying on the production server, we'll want to simplify our lives and have a single npm script to:
  1. pull
  2. build
  3. start app

- On your personal machine, add the following script to package.json's scripts:

  - "prod": "git reset --hard && git pull && npm run build && sudo NODE_ENV=production npm run start"

- You'll need to push this to your repo and then pull while logged into the server.

Register as Systemd Service

- Edit Systemd Configuration file For the App

```
$ sudo nano /lib/systemd/system/blog_app.service
```

```
[Unit]
Description=Travel Blog
After=network.target

[Service]
Environment=NODE_ENV=production
Type=simple
User=ubuntu
WorkingDirectory=/home/ubuntu/travel-notes-<YourGithubName>
ExecStart=/usr/bin/npm run prod
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- Enable the service: $ sudo systemctl enable blog_app.service
- Run the service: $ sudo service blog_app start

Add a Deploy Script

- SSH'ing in and running commands manually is tedious and error prone
- Let's add one more npm script to make deploying a two step process of:
  - Commit and Push your Changes in git
  - $ npm run deploy
- We can achieve this by using SSH in a "command mode" to run the command on our AWS server that restarts your app (replace <ip> with your AWS server's IP:
  - "deploy":
    "ssh -i ~/.ssh/COMP426-2019.pem ubuntu@<ip> 'sudo service blog_app restart'"

Listening on the correct Port

- In production (for now) we'd like to listen on the default HTTP port 80
  - Ultimately, we'd like to get HTTPS going but that'll take more admin steps
- To do so, we'll look for a special environment variable to tell us if our server is running in production mode (or not).

```
const PORT = process.env.NODE_ENV === "production" ? 80 : 1234;
app.listen(PORT, () => console.log(`Listening on ${PORT}`))
    .on("error", (e) => console.error(e));
```